# How to run simulation for the UDP Transmitter Module

The testbench is designed to read input data from text file and write the output of the module to text file. Each input text file represents test scenario for the transmitter module. The steps to run any scenario are as follow:

1. Download udp_tx.xpr.zip from Github.
2. Extract the project from the zip file.
3. Open udp_tx.xpr project in Vivado 2016.4.
4. In the sources window expand Simulation Sources, then expand sim_1, set either udp_tx_tb or udp_tx_tb_post_synth as top.
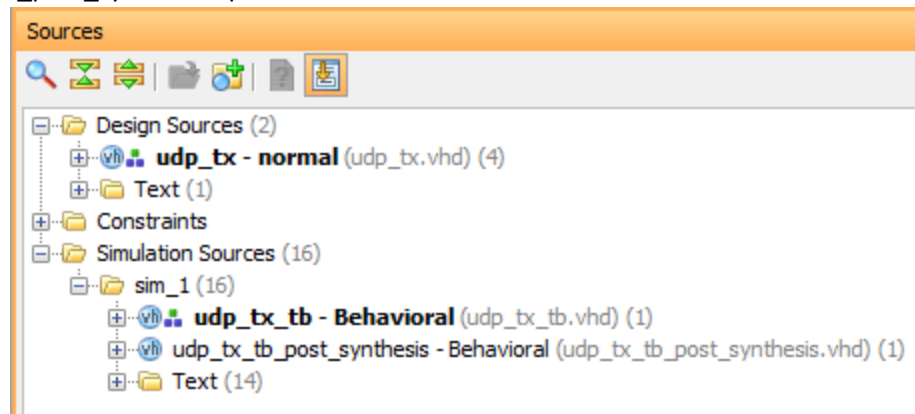
5. Open the simulation source file.
6. Scroll down in the testbench to line 64 to change the input file name to which text file you are going to use in the simulation.

```
file In_file : text open read_mode is "zero-length.txt";-- Change the file name
```

7. Scroll down to line 89 to change the number of udp packets will be tested based on the provided table.
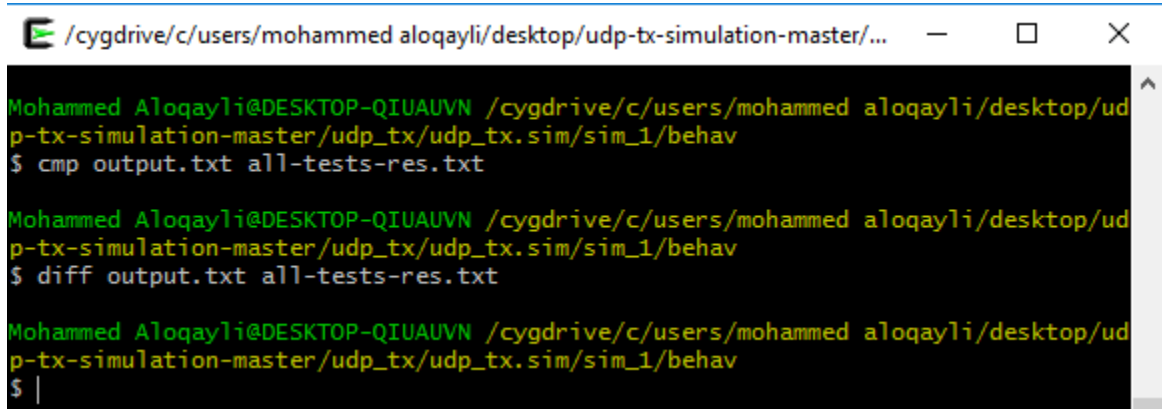
```
signal Num_of_pckts : POSITIVE := 1;
```

Text File	Number of UDP Packets
zero-length.txt	1
odd-length1.txt	1
odd-length2.txt	1
even-length.txt	1
Max-length.txt	1
consecutive-packets.txt	3
all-tests.txt	14

8. Run behavioral/post-synthesis simulation for at least 500 ns except for all-tests, you need to run the simulation 10us.
9. Then the testbench will write the output of the module to a text file "output.txt".

```
file Out_file : text open write_mode is "output.txt";
```

10. In order to check whether the module is behaving as expected or not, you need to compare between "output.txt" file and expected result text file (i.e. zero length test case will have expected result in "zero-length-res.txt").

11. To compare between text files, you need to use cmp or diff tools. For Windows, by using **Cygwin** you can utilize those tools. If the files are matching nothing will be returned in the prompt.
Note: Both files will be in the simulation folder (i.e. … udp_tx\udp_tx.sim\sim_1\behav or udp_tx\udp_tx.sim\sim_1\synth\timing).
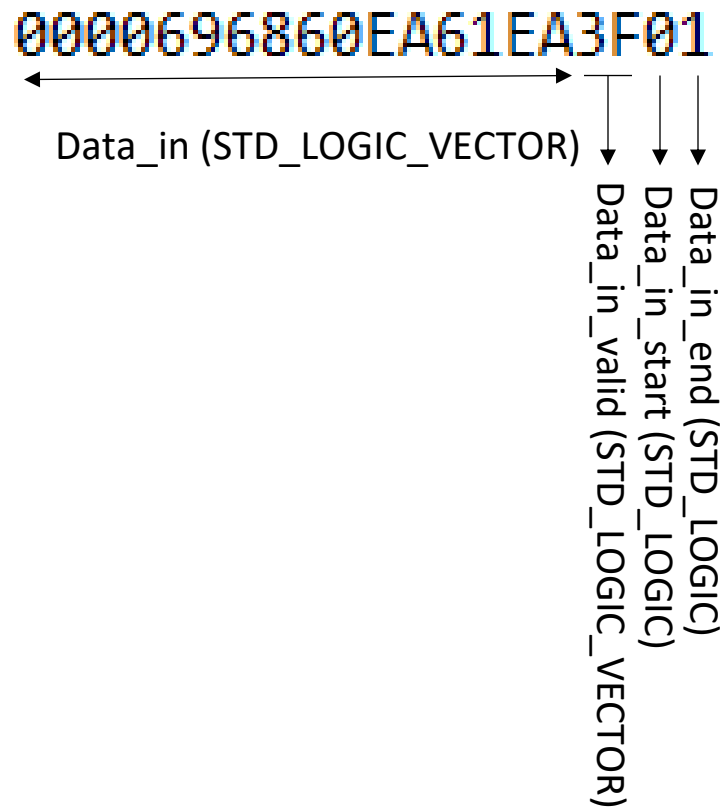
Test Data Format:

Each input text file is arranged as follow:

- First 8 bytes represent data in hexadecimal.
- Next 2 bytes represent valid signal in hexadecimal.
- Next bit represents start signal in binary.
- Last bit represents end signal in binary.

0000696860EA61EA3F01

Data_in (STD_LOGIC_VECTOR)

Data_in_valid (STD_LOGIC_VECTOR)

Data_in_start (STD_LOGIC)

Data_in_end (STD_LOGIC)

Written by: Mohammed Aloqayli

4/18/17